MandelbrotApp.java

```java
package oop.mandelbrot;

import java.awt.EventQueue;

import javax.swing.JFrame;

/**
 * Die Klasse MandelbrotApp ist die Hauptklasse der Mandelbrot-Anwendung.
 * Sie implementiert das Runnable-Interface, um die Anwendung im Swing-Event-Dispatch-Thread
 * auszuführen.
 *
 * @author Aleksandar Travanov
 * @version 1.0
 */
public class MandelbrotApp implements Runnable {

    /**
     * Die main-Methode, die die Anwendung startet.
     * @param args Die Befehlszeilenargumente (werden nicht verwendet)
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new MandelbrotApp());
    }

    /**
     * Führt die Mandelbrot-Anwendung im Swing-Event-Dispatch-Thread aus.
     */
    @Override
    public void run() {
        // Erzeugt ein MandelbrotFrame-Objekt mit den angegebenen Parametern
        MandelbrotFrame frame = new MandelbrotFrame(532, -2, 2, 100);
        // Setzt den Titel des Frames
        frame.setTitle("Mandelbrot Menge");
        // Setzt das Schließen-Verhalten des Frames
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Macht den Frame sichtbar
        frame.setVisible(true);
    }
}
```

package-info.java

```java
/**
 * Das Paket oop.mandelbrot enthält Klassen zur Erzeugung und Darstellung des Mandelbrot-Fraktals.
 *
 * @author Aleksandar Travanov
 * @version 1.0
 */

package oop.mandelbrot;
```

MandelbrotFrame.java

```java
package oop.mandelbrot;

import java.awt.image.BufferedImage;
import java.awt.image.WritableRaster;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

import oop.mandelbrot.coordinate.CrazyCoordinate;

/**
 * Die Klasse MandelbrotFrame erzeugt ein JFrame-Fenster,
 * das ein Bild des Mandelbrot-Fraktals darstellt.
 *
 * @author Aleksandar Travanov
 * @version 1.0
 */
@SuppressWarnings("serial")
public class MandelbrotFrame extends JFrame {
    /**
     * Konstruiert ein neues MandelbrotFrame-Objekt mit der angegebenen Bildgröße,
     * Bereich und Anzahl von Iterationen.
     *
     * @param sizeImg Die Größe des Bildes (Breite und Höhe)
     * @param from Der Startwert des Bereichs für Real- und Imaginärteile
     * @param to Der Endwert des Bereichs für Real- und Imaginärteile
     * @param nofIterations Die Anzahl von Iterationen für die Mandelbrotberechnung
     */
    public MandelbrotFrame(int sizeImg, double from, double to, int nofIterations) {
        BufferedImage image = createMandelbrotImg(sizeImg, from, to, nofIterations);
        add(new JLabel(new ImageIcon(image)));
        pack();
    }

    /**
     * Erzeugt ein BufferedImage, das das Mandelbrot-Fraktal darstellt.
     * Dabei wird für jeden Punkt im Bildbereich die Mandelbrot-Funktion iterativ berechnet,
     * um festzustellen, ob der Punkt zur Mandelbrot-Menge gehört oder nicht.
     *
     * @param sizeImg          Die Größe des Bildes (Breite und Höhe)
     * @param from             Der Startwert des Bereichs für Real- und Imaginärteile
     * @param to               Der Endwert des Bereichs für Real- und Imaginärteile
     * @param nofIterations    Die Anzahl von Iterationen für die Mandelbrotberechnung
     * @return                 Das erzeugte BufferedImage, das das Mandelbrot-Fraktal darstellt
     */
    private BufferedImage createMandelbrotImg(int sizeImg, double from, double to, int
nofIterations) {
        double size = to - from;
        BufferedImage image = new BufferedImage(sizeImg, sizeImg, BufferedImage.TYPE_INT_RGB);
        WritableRaster raster = image.getRaster();
        int[] blackPixel = { 0, 0, 0 };
        for (int i = 0; i < sizeImg; i++) {
            for (int j = 0; j < sizeImg; j++) {
                double x = from + (i * size) / sizeImg;
                double y = from + (j * size) / sizeImg;
                CrazyCoordinate z = new CrazyCoordinate(0.0, 0.0);
                CrazyCoordinate c = new CrazyCoordinate(x, y);
                int counter = 0;
                do {
                    counter++;
                    z = z.mul(z).add(c);
                } while ((counter < nofIterations) && z.scalar() <= (to - from));

                if (counter == nofIterations) {
                    raster.setPixel(i, j, blackPixel);
                } else if (counter < nofIterations / 10) {
                    int[] bluePixel = { 0, 0, (counter*300)%255 };
                    raster.setPixel(i, j, bluePixel);
                } else {
                    int[] redPixel = {255,0,0 };
                    raster.setPixel(i, j, redPixel);
                }
            }
        }
        return image;
    }
}
```

CrazyCoordinate.java

```java
/**
 *
 * @author Kristian Lubina
 * @version 1.0
 */

package oop.mandelbrot.coordinate;

/**
 * Die Klasse CrazyCoordinate repräsentiert eine Koordinate in der komplexen Ebene.
 */

public class CrazyCoordinate {
                        //Properties
    private double x;
    private double y;

    /**
     * Konstruktor, der den Realteil (x) und den Imaginärteil (y) einer komplexen Zahl initialisiert.
     * @param x Der Realteil der komplexen Zahl.
     * @param y Der Imaginärteil der komplexen Zahl.
     */

    public CrazyCoordinate(double x, double y) {
        this.x = x;
        this.y = y;
    }

    /**
     * Addiert diese Koordinate mit einer anderen und gibt das Ergebnis zurück.
     * @param other Die zu addierende Koordinate.
     * @return Eine neue Koordinate, die das Ergebnis der Addition ist.
     */

    public CrazyCoordinate add(CrazyCoordinate other) {
        return new CrazyCoordinate(this.x + other.x, this.y + other.y);
    }

    /**
     * Multipliziert diese Koordinate mit einer anderen und gibt das Ergebnis zurück.
     * @param other Die zu multiplizierende Koordinate.
     * @return Eine neue Koordinate, die das Ergebnis der Multiplikation ist.
     */

    public CrazyCoordinate mul(CrazyCoordinate other) {
        return new CrazyCoordinate(this.x * other.x - this.y * other.y, this.x * other.y + this.y * other.x);
    }

    /**
     * Gibt den Realteil der Koordinate zurück.
     * @return Der Realteil der Koordinate.
     */

    public double getx() {
        return this.x;
    }

    /**
     * Gibt den Imaginärteil der Koordinate zurück.
     * @return Der Imaginärteil der Koordinate.
     */

    public double gety() {
        return this.y;
    }

    /**
     * Berechnet das skalare Produkt dieser Koordinate mit sich selbst.
     * @return Das skalare Produkt der Koordinate.
     */

    public double scalar() {
        return this.x * this.x + this.y * this.y;
    }

    /**
     * Gibt die Koordinate in der String-Form zurück.
     * @return Die String-Repräsentation der Koordinate.
     */

    @Override
    public String toString() {
        return "(" + this.x + ", " + this.y + ")";
    }
}
```